THE UNIVERSITY of EDINBURGH

Edinburgh Research Explorer

ProPPA: Probabilistic Programming for Stochastic Dynamical Systems

Citation for published version:
Georgoulas, A, Hillston, J & Sanguinetti, G 2018, 'ProPPA: Probabilistic Programming for Stochastic Dynamical Systems', *ACM Transactions on Modeling and Computer Simulation*, vol. 28, no. 1, 3. https://doi.org/10.1145/3154392

Digital Object Identifier (DOI):
10.1145/3154392

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Peer reviewed version

Published In:
ACM Transactions on Modeling and Computer Simulation

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 05. May. 2023

ProPPA: Probabilistic Programming for Stochastic Dynamical Systems

ANASTASIS GEORGOULAS, University College London
JANE HILLSTON, University of Edinburgh
GUIDO SANGUINETTI, University of Edinburgh

Formal languages like process algebras have been shown to be effective tools in modelling a wide range of dynamic systems, providing a high-level description that is readily transformed into an executable model. However their application is sometimes hampered because the quantitative details of many real-world systems of interest are not fully known. In contrast, in machine learning there has been work to develop probabilistic programming languages, which provide system descriptions that incorporate uncertainty and leverage advanced statistical techniques to infer unknown parameters from observed data. Unfortunately current probabilistic programming languages are typically too low-level to be suitable for complex modelling.

In this paper we present ProPPA, the first instance of the probabilistic programming paradigm being applied to a high-level, formal language, and its supporting tool suite. We explain the semantics of the language in terms of a quantitative generalisation of Constraint Markov Chains and describe the implementation of the language, discussing in some detail the different inference algorithms available, and their domain of applicability. We conclude by illustrating the use of the language on simple but non-trivial case studies: here ProPPA is shown to combine the elegance and simplicity of high-level formal modelling languages with an effective way of incorporating data, making it a promising tool for modelling studies.

CCS Concepts: • **Computing methodologies** → **Modeling methodologies**; **Uncertainty quantification**; • **Mathematics of computing** → **Markov processes**; *Bayesian computation*; *Markov-chain Monte Carlo methods*;

General Terms: Terms

Additional Key Words and Phrases: process algebra, stochastic modelling, probabilistic programming, parameter estimation

ACM Reference format:
Anastasis Georgoulas, Jane Hillston, and Guido Sanguinetti. 2017. ProPPA: Probabilistic Programming for Stochastic Dynamical Systems. *ACM Trans. Model. Comput. Simul.* 1, 1, Article 1 (January 2017), 24 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Stochastic process algebras are an established tool for modelling and analysing the behaviour of dynamical systems, combining theoretical elegance with a range of attractive and practically useful features — compositionality, formal interpretation of models and the ability to verify their behaviour using model-checking, to list a few. The starting point for process algebras, as for many other formal modelling methods, is a full specification of the system being modelled, both in terms of interaction structure and of parameters quantifying the (infinitesimal) dynamics of the system.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2017 ACM. XXXX-XXXX/2017/1-ART1 $15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

The formal defined semantics of the language allows the high-level description to be automatically mapped to an underlying mathematical model suitable for analysis.

In many application fields, however, prior knowledge of systems is usually incomplete: even when the structure of interactions is known, the quantitative details seldom are. This is particularly apparent in fields like systems biology: even though there has been widespread interest in formal modelling of biochemical processes, parameters such as reaction rates are difficult to measure experimentally, leading to incomplete models. This presents a stumbling block to the direct and efficient application of such modelling formalisms, thus both depriving the field of a formal approach, and restricting the possible application domains of formal frameworks.

To deal with this incomplete specification, once a model has been formulated, parameters are often tuned so as to match the observed data or a subset thereof. This search, even when guided by expert knowledge in the field about likely values, is a time-intensive process and, when performed manually, may be error-prone. The recognition of these difficulties has led to some work (such as [31, 33]) on automating this procedure. However, proposed frameworks employ simple greedy algorithms for obtaining a good estimate. This implies an assumption that a single solution to the problem exists, and that the observed data is sufficient to accurately retrieve it — a position that is hard to justify, particularly in the presence of noise and stochastic dynamics. Additionally, by simply locating a good parameter estimate, these methods do not provide a statistical framework for quantifying the uncertainty in their predictions.

On the other hand, in the field of *machine learning*, a larger body of research ([9, 39, 42], among others) has proposed methods for inferring the parameters of stochastic systems from measured data. These methods are statistically rigorous and provide richer results in the form of a distribution over possible values. However, that work has focused on low-level descriptions of the system; therefore, in order to use them, one has to forego the attractive features of high-level languages and their advantages.

In order to facilitate the use of such inference methods, various frameworks have been proposed following the *probabilistic programming* paradigm, in which the user can perform *inference by programming*: a high-level language is used to describe a probabilistic model, including prior beliefs about any uncertain quantities, and specifying observed data. Based on this description, appropriate inference algorithms are automatically instantiated and executed, providing a distribution for the parameters that is consistent with the model and the observations. However, current frameworks such as [23, 36, 40]) resemble conventional programming languages more than modelling ones: complex dynamics, such as those of stochastic systems, need to be written out explicitly, and their inference capabilities are limited when it comes to dynamical models. A recent survey of the field [26] highlighted the difficulties associated with modelling time, and concluded that existing frameworks are not sufficient in their treatment of dynamical systems.

In this paper we present work that aims to address these problems by introducing a formal modelling language which directly incorporates observations (and the associated uncertainty) and can leverage cutting-edge statistical machine learning tools to perform inference and quantify uncertainty. We present ProPPA, a Probabilistic Programming Process Algebra; to our knowledge, this is the first example of the probabilistic programming paradigm being extended to a higher-level, formal system description language like a process algebra. Note that the ability to perform inference from data qualitatively distinguishes our approach from general stochastic modelling methodologies such as stochastic process algebras, which simply incorporate uncertainty in model evolution through the use of random variables to determine rates.

ProPPA was first presented in [18] as a process algebra with basic inference capabilities. Here we recall the syntax and semantics of the language and give a detailed account of the sophisticated

inference engine supports it, bringing ProPPA to the level of a complete probabilistic programming framework. This allows the modeller to select from a range of algorithms without having to provide their own implementation. A number of examples are considered, including the use of a continuous approximation of models written in the language.

ProPPA is based on the stochastic process algebra Bio-PEPA [13], and inherits many of its qualities. We show how to include uncertainty in the definition of the language (Section 3), and propose an appropriate semantic model for uncertain models (Section 4). We adopt a modular approach to construct our language, so that the core language is capable of adopting different machine learning methodologies (Section 5) to perform inference from possibly very different types of data. We demonstrate the power of this approach by performing inference in different examples in Section 6.

1.1 Related work

Although parameter fitting of formal models is often done manually, some previous work has explored automated parameter estimation methods for such models. The Evolving Process Algebra (EPA) framework of Marco *et al.* was introduced in [31] to optimize the parameters of models written in the PEPA language and uses genetic algorithms to find parameters that make the model match some observed behaviour. Good parameters are found by simulating the model using those values multiple times, taking the average of these traces and then comparing the average trace with the observed one, using some distance metric. The framework can also optimize parameters of Bio-PEPA models and has been applied to a model of immune response [32]. The work in [47] looks at parameter optimisation for models written in the Calculus of Wrapped Components, a language particularly suited to describing biological systems with locations. This approach searches for a good solution using constraint optimization methods, claimed to be less computationally intensive than evolutionary computation approaches like genetic algorithms. The problem has also been considered in the case of BIOCHAM [12], initially by exhaustively searching the space of possible parameters to find good values — in this case, ones that make the system satisfy a set of temporal logic properties describing desired behaviour. Subsequent work [43, 44] has proposed more sophisticated search methods by combining evolutionary computation with a continuous degree of property satisfaction.

The capacity planning tool of [53] addresses a similar problem. The tool works with PEPA models and its goal is to optimize the size of a given system. More precisely, assuming that every resource has an associated cost, its goal is to find a configuration of components that minimizes the total cost while still satisfying some user-specified performance constraints. The search over configurations is performed using particle swarm optimization.

Going beyond parameter estimation, a harder task is that of synthesising a whole model based on some observed data. This involves not only the numerical parameters but also the *structure* of the model itself. The work in [45] examined the use of genetic programming to tackle this problem for a subset of the stochastic π-calculus. The EPA framework includes similar functionality for Bio-PEPA models [33], also using genetic programming to perform a heuristic search over part of the definitions in the model.

A somewhat related problem for partially-specified formal models is treated in [5]: there, the focus is not on fitting uncertain parameters but on using symbolic analysis to reason about the behaviour of the unspecified components. As an alternative to parameter estimation, Brim *et al.* [10] consider calculating the satisfaction probability of a formula over an entire parameter space, and propose an approximate way of performing this computation.

With regards to probabilistic programming, as mentioned previously, current languages are not well-suited to modelling complex dynamical systems. An initial attempt to apply probabilistic programming to continuous-time models of biological systems was developed in [19] based on the Infer.NET Fun language [6]. Despite using a lower-level description of CTMCs, the built-in inference engine of Fun could not cope with the continuous-time dynamics. Performing inference therefore required the development of a new algorithm, highlighting the limitations of the existing language.

2 BACKGROUND

This section gives some information on the language on which ProPPA is based, the frameworks and mathematical objects used for the definition of its semantics, and the field of probabilistic programming from which we draw inspiration.

2.1 Process algebras and Bio-PEPA

Process algebras are a family of languages first used to model concurrent systems, by specifying the system's components and the actions that these may perform. The formal nature of the languages allow one to reason about the behaviour of the modelled system, such as verifying that undesirable states (configurations of the system) are avoided or that simple properties hold. The sub-family of stochastic process algebras (e.g. PEPA [28], IMC [27], EMPA [1]) extend this framework by introducing time into the system and assuming that the time for a transition to occur is an exponentially distributed random variable. The parameter of the distribution is called the rate of the transition, and, when multiple transitions are possible, the probability of choosing a particular transition is proportional to its rate. These high-level description languages are then mapped to a Continuous Time Markov Chain (CTMC) via a formal semantics based on a labelled transition system, which captures all the possible states of the system and the transitions between them. CTMCs can be analysed in a number of ways, a very common one being simulation via Gillespie's Stochastic Simulation Algorithm (SSA) [22], which draws a trace from a fully specified CTMC (i.e. one in which all the transition rates can be calculated).

The description of a system in a process algebra can be used to implicitly generate its state space, in the form of a labelled transition system (LTS). A LTS is a graph whose nodes are the system's states and whose edges are the possible transitions between states, labelled with some information (e.g. what reaction causes the transition or at what rate the transition occurs). Analysing the LTS can give important insights into the behaviour of the system, such as whether a state is reachable under certain conditions or within a specified time frame. Verifying whether a system description satisfies such properties is the subject of model checking algorithms and tools, with the properties to be checked often being expressed in a temporal logic, such as CSL [4] or CTL [14].

Bio-PEPA [13] is a stochastic process algebra based on PEPA but designed for the modelling of biological processes. In Bio-PEPA, system components (termed *species*) are defined through their behaviour, that is, how they interact with each other, reflecting a reagent-centric modelling style. The definition of a species takes the form

$$A = (\alpha_1, k_1)op_1 + \cdots + (\alpha_n, k_n)op_n \text{ where } op_i = \downarrow, \uparrow, \oplus, \ominus \text{ or } \odot$$

which means that species A takes part in reaction α_i with stoichiometry k_i (a stoichiometry of one can be omitted for simplicity). The different options for op_i correspond to different roles of A in α_i: reactant, product, catalyst, inhibitor or generic modifier, respectively. These definitions are composed using the choice operator (+) to describe species that can take part in multiple reactions.

Each reaction has an associated rate law, which can be specified either as a formula or using a predefined law (such as mass-action or Hill kinetics). Parameters can be defined and used, for example, in kinetic laws or as initial concentrations, but their values must be specified and are considered fixed. The language results in a modular or compositional approach, wherein the behaviour of the system emerges as a direct consequence of the behaviour of the species (without the need to, for instance, explicitly write out ODEs or chemical equations for reactions, as they can be automatically computed). This means modifications to the model can be performed by changes to the "local" species definitions.

Formally, a Bio-PEPA system is defined as a tuple $\langle \mathcal{V}, \mathcal{N}, \mathcal{K}, \mathcal{F}_R, Comp, P \rangle$, where $\mathcal{V}$ is the set of compartments (locations) in the system; $\mathcal{N}$ is a set of quantities associated with each species, such as its maximum concentration; $\mathcal{K}$ is the set of parameters; $\mathcal{F}_R$ is the set of rate laws; $Comp$ is the set of sequential components (species definitions); and P is the model component, which describes how the various species cooperate with each other as well as their initial concentrations. An example of a model component comprising three species A_i with initial quantities l_i is

$$A_1[l_1] \underset{*}{\bowtie} A_2[l_2] \underset{*}{\bowtie} A_3[l_3]$$

Although originally designed for modelling biological processes, Bio-PEPA has found application in other domains of large scale collective behaviour, such as crowd movement [35] and urban transport [52].

2.2 Probabilistic Programming

Probabilistic programming is a framework for reasoning about uncertain processes in a statistically consistent manner. In a probabilistic program, uncertain aspects of the system, such as unknown parameters, are treated as random variables and can be assigned probability distributions that express this uncertainty. Additionally, one can specify observations of the system, from which information about the unobserved aspects can be gleaned. In other words, the program specifies a probability distribution, which can be viewed in two ways: one can sample from it, essentially simulating the system; or, if one has additional information about the system, one can condition the distribution on this data, inferring an updated distribution over the unknown variables that takes into account this new knowledge. Probabilistic programming offers an elegant approach for treating uncertain systems in these two ways, automating the process to a degree and eliminating the need for bespoke inference solutions, as the inference algorithm can be configured and executed automatically based on the structure of the program.

Previous work has focused mainly on integrating the paradigm into traditional programming languages, giving rise to frameworks like Church [23], IBAL [40] and Infer.NET [36] (along with its interfaces Fun [6] and Tabular [25]), as well as more recent languages such as R2 [38] and WebPPL [24]. These frameworks differ in the inference methods they offer, but they all describe systems at a low level: one must explicitly specify all the statistical dependences between the different variables, yielding potentially large descriptions which are difficult to manage. This limits the range of systems that can be modelled, with continuous-time dynamical systems being particularly hard or even impossible to deal with. A language for describing continuous-time systems has been proposed in [41], but still lacks the formal features of a process algebra and an available implementation. The PRISM language [46], inspired by logic programming, presents an interesting direction: it provides a more formal style for encoding probabilistic models and, through recursion, a class of stochastic processes. Although learning methods have been proposed for such programs [2], there is no explicit concept of continuous time. In light of these shortcomings, we advocate combining the principle of probabilistic programming with a formal language like

a process algebra, for a flexible, high-level framework in which to model and analyse complex systems with uncertain aspects.

3 A PROBABILISTIC PROGRAMMING PROCESS ALGEBRA

The ProPPA syntax is based on Bio-PEPA, with the addition of two key features that introduce aspects of probabilistic programming. The first concerns the representation of uncertainty in the system. We should note that we are only considering uncertainty in the kinetics, and assume that we fully know what reactions each species can take part in. In Bio-PEPA, parameters can be used in the definition of kinetic rate functions, but their values must be fixed. With this in mind, we allow uncertain parameters, whose values are given as probability distributions rather than concrete numbers. The second feature is a way of incorporating information about the behaviour of the system into the model: these will be the *observations*.

For the purposes of this paper, we consider observations in the form of a time-series, i.e. a sequence of (possibly noisy) measurements of the state of the system at a finite number of points. However, the observations could more generally be any observed function of the specific trajectory of the system (specified through a temporal logic formula, for instance) — we return to this point in Section 7. Even though we use the biological paradigm "inherited" from Bio-PEPA for the rest of the paper, especially with respect to terminology, we stress that this modelling approach can be applied to a much broader set of applications.

On a formal level, we will need to modify the definition of a Bio-PEPA system (given previously in Section 2.1), mainly by reconsidering the role of the set of parameters $\mathcal{K}$. We extend the system definition in two ways, corresponding to the two features described above. Firstly, since the uncertain quantities are represented as parameters in the model, we extend the definition of a parameter to include a distribution rather than a concrete value. These are the *prior distributions* or *priors* over parameters, which express our belief about a parameter's values before seeing any data. The set of parameters $\mathcal{K}$ is then partitioned into two subsets: $\mathcal{K}_c$ comprises the concrete parameters, while $\mathcal{K}_u$ contains the uncertain ones, along with the priors associated with them. We write $(k \sim \mu) \in \mathcal{K}_u$ if the parameter k is drawn *a priori* from the distribution μ. Importantly, the functional rates $\mathcal{F}_R$ can refer to any parameter in $\mathcal{K}_u$ as well as those in $\mathcal{K}_c$; in this sense, a functional rate can represent a family of functions.

Secondly, we add a new component $\mathcal{O}$ representing the observations. These impose restrictions on the acceptable parameter values and modify our belief about their distribution, as described in more detail in Section 5.1. Extending the syntax to accommodate these features is straightforward. A ProPPA system is therefore a tuple $\langle Comp, \mathcal{K}_c, \mathcal{K}_u, \mathcal{F}_R, \mathcal{O}, P\rangle$, with the other components retaining the meaning they have in Bio-PEPA. Following the terminology of [17], we will also write a system as $\langle \mathcal{T}, P\rangle$ where $\mathcal{T} = \langle Comp, \mathcal{K}_c, \mathcal{K}_u, \mathcal{F}, \mathcal{O}\rangle$ is called the context.

3.1 Syntax

The text of a ProPPA model has five parts: parameter definitions; reaction definitions; species definitions; initial state; and the inference specification.

$$
\begin{aligned}
model ::= \ & parameter\_def\_list \\
& react\_def\_list \\
& species\_def\_list \\
& init\_state \\
& infer\_spec
\end{aligned}
$$

where *X_def_list* indicates a list of *X_def* definitions. The following grammar defines each of these parts.

$$
\begin{aligned}
\mathit{parameter\_def} &::= \mathit{num\_value} \\
&\quad \mid \mathit{distr\_def} \\
\mathit{react\_def} &::= \texttt{kineticLawOf}\ \mathit{react}: \ \mathit{function\_def} \\
\mathit{species\_def} &::= \mathit{atomic\_def} \\
&\quad \mid \mathit{atomic\_def} + \mathit{species\_def} \\
\mathit{atomic\_def} &::= (\mathit{react}, n)\mathit{op} \\
\mathit{op} &::= \uparrow \mid \downarrow \mid \oplus \mid \ominus \mid \odot \\
\mathit{init\_state} &::= \mathit{comp}[n] \\
&\quad \mid \mathit{comp}[n] \underset{\mathcal{L}}{\bowtie} \mathit{init\_state} \\
\mathit{infer\_spec} &::= \mathit{observe\_def}; \mathit{infer\_def}; [\mathit{conf\_def}] \\
\mathit{observe\_def} &::= \texttt{observe}(\mathit{file}) \\
\mathit{infer\_def} &::= \texttt{infer}(\mathit{algorithm}) \\
\mathit{conf\_def} &::= \texttt{configure}(\mathit{file})
\end{aligned}
$$

where $\mathit{num\_value} \in \mathbb{R}$, *distr_def* is a probability distribution with its associated parameters (e.g. $\mathit{Gaussian}(0, 1)$), *react* is a reaction name, *function_def* is a function definition, $n \in \mathbb{N}^*$, $\mathcal{L}$ is a set of reaction names, *file* is a file name and *algorithm* is the name of an inference algorithm (Section 5.3).

3.2 A rumour spreading example

As an example of a ProPPA model, we will consider a population CTMC model of rumour spreading over a network [15]; this consists of three types of agents (Figure 1). A spreader (S) is someone who has already encountered the rumour and is actively trying to spread it. When an ignorant (I) meets a spreader, the ignorant also becomes a spreader. When two spreaders meet, one of them stops spreading and becomes a blocker (R), reflecting the idea that only new rumours are worth spreading. A blocker can then convert spreaders into other blockers. The dynamics of the system can exhibit qualitatively different behaviours depending on the parameter values: in particular, two possible steady state regimes exist, where all agents are in blocker state, or where a blocker and an ignorant population coexist.

Figure 2 shows the description of the system in ProPPA for an initial population of 15 agents. The behaviour of the different agents is described in lines 6-8: line 6 says that the count of ignorants is decreased by 1 when the `spread` interaction occurs, and the other types of agents are similarly defined by the changes to their count through the various interactions. Line 9 shows the initial

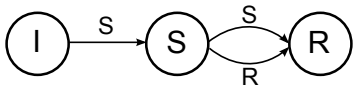

Fig. 1. State transitions of a rumour-spreading agent. The arrow labels indicate the type of agent that must be encountered for the corresponding transition to take place.

```
1  ( 1)  k_s = Uniform(0,1);
2  ( 2)  k_r = Uniform(0,1);
3
4  ( 3)  kineticLawOf spread : k_s * I * S;
5  ( 4)  kineticLawOf stop1 : k_r * S * S;
6  ( 5)  kineticLawOf stop2 : k_r * S * R;
7
8  ( 6)  I = (spread,1) ↓ ;
9  ( 7)  S = (spread,1) ↑ + (stop1,1) ↓ + (stop2,1) ↓ ;
10 ( 8)  R = (stop1,1) ↑ + (stop2,1) ↑ ;
11
12 ( 9)  I[10] ⋈* S[5] ⋈* R[0]
13
14 (10)  observe(obs_rumour);
15 (11)  infer(ABC);
```

Fig. 2. ProPPA model of the rumour spreading example.

population of each kind of agent; the cooperation $\underset{*}{\bowtie}$ means that the agents synchronise on all shared reactions. We assume that these interactions happen at rates that obey mass-action kinetics, i.e. they are proportional to the count of the agents involved. We also assume that the rate constants for the spreader-spreader and spreader-blocker interactions are the same (k_r), while the spreader-ignorant interaction has a rate constant k_s; this is shown in lines 3-5. The definition of k_s and k_r as uniformly distributed (lines 1-2) reflects our prior belief that, without seeing any data, they are not biased towards any value in their domain, which in this case we chose to be $[0, 1]$. We will use this model as a running example for the rest of this paper; lines 10-11 are discussed in Section 5.1.

4 PROPPA SEMANTICS

As mentioned earlier, Bio-PEPA models can be mapped to CTMCs. In a CTMC, every transition between states has a concrete rate, making this interpretation unsuitable for a language with uncertainty, such as ProPPA. We must therefore use a different object to define the semantics of our language, one that is more suited to describing uncertain models, such as Constraint Markov Chains (CMC, [11]). These are a generalisation of Discrete Time Markov Chains in which the probability of transitioning from a given state to another does not have a fixed value. Instead, the CMC specifies a constraint that the values of the various transition probabilities must obey or, equivalently, a set of acceptable values for them.

The way CMCs are defined presents two limitations from our perspective. Firstly, they have been defined only for discrete-time systems, whereas we are interested in modelling in continuous time. Their definition can be adapted to the continuous-time domain through simple alterations as shown in the definition below. Secondly, while a CMC defines the set of possible values for a rate, it gives no information on the relative likelihood of those values. For our purposes we must move from this purely non-deterministic setting to a probabilistic setting, where, instead of a binary decision, we have quantitative information about our belief in the plausibility of a value. This leads to the definition of a Probabilistic Constraint Markov Chain (PCMC) [18].

Definition 4.1. A Probabilistic Constraint Markov Chain is a tuple $\langle S, o, A, V, \phi \rangle$, where:

- S is the set of states.
- $o \in S$ is the initial state.
- A is a set of atomic propositions.
- $V : S \to 2^{2^A}$ gives a set of acceptable labellings for each state.

- $\phi : S \times S \times [0, \infty) \to [0, \infty)$ is the *constraint function.*

The constraints in a PCMC are on rates rather than transition probabilities, reflecting the shift to continuous time. Additionally, ϕ now describes a probability density function (pdf), therefore it takes values in $\mathbb{R}_+$ instead of $\{0, 1\}$, with the additional restriction that $\int_0^\infty \phi(s, s', r)\mathrm{d}r = 1$ for any $s, s' \in S$. The resulting object is richer than a CMC, and the additional information it can capture means that PCMCs are ideally suited to define the semantics of ProPPA models.

Following the standard approach of stochastic process algebras the PCMC is derived via a formal semantics in terms of transition relations that then give rise to a labelled transition system. As with Bio-PEPA the semantics is given in terms of two relations. The *capability relation* describes what transitions may occur between states, without giving any quantitative information about the rates — in other words, it gives the structure of the transition system. The *stochastic relation* uses that information, as well as the definition of the kinetic functions, to provide the rates of the transitions, thus completing the labelling of the transition system. Since there is no uncertainty in the qualitative behaviour of the species, the capability relation for ProPPA remains unchanged from that for Bio-PEPA. The uncertainty, as a pdf over rate values, is captured in the stochastic relation.

As described earlier, a ProPPA model corresponds to a whole family of systems, each of them generated by a particular assignment of values to the uncertain parameters. Furthermore, these systems are weighted, in that there is a probability distribution over them. Essentially, the semantics provides the means to translate the uncertainty over the parameters into uncertainty over systems. Constructing the capability and stochastic relations is the first step towards this: the semantics describes how the pdf over parameters induces a pdf over values of transition rates. In a second step, the resulting distributions are used to define a PCMC, which captures the set of possible concrete systems. The full definition of the transition relations was first given in [18] and, for completeness, is also presented in Appendix A. The rest of this section touches on some points of particular importance.

The crucial point, as mentioned above, is propagating the uncertainty over parameters so as to calculate the pdf over values of transition rates. This is encapsulated in the stochastic relation, and the main idea is to consider the rates as functions of parameters.

Assume the rate Y of a reaction depends on a parameter Θ and let $Y = T(\Theta)$ express this dependence. We know, from the context, that Θ is distributed according to a probability density function (pdf) f_Θ. Y, being a transformation of Θ, will also follow a distribution, whose pdf we denote f_Y. If the function T is strictly monotonic, f_Y can be obtained through a simple change of variable:

$$f_Y(y) = \left|\frac{dT^{-1}(Y)}{dY}\right|_{Y=y} f_\Theta(T^{-1}(y)) \tag{1}$$

where $T^{-1}(y)$ is the (unique) value of the parameter Θ for which the rate is y.

To see why this is valid, let us first consider the case where T is strictly increasing, which implies that the inverse function T^{-1} is well-defined and is also increasing. The cumulative distribution function of Y is then:

$$F_Y(y) = P\left(Y \leq y\right) = P\left(T(\Theta) \leq y\right) = P\left(\Theta \leq T^{-1}(y)\right) = F_\Theta\left(T^{-1}(y)\right)$$

and the corresponding pdf is:

$$f_Y(y) = \left.\frac{dF_Y(y)}{dY}\right|_{Y=y} = \left.\frac{dF_\Theta\left(T^{-1}(Y)\right)}{dY}\right|_{Y=y} = f_\Theta\left(T^{-1}(y)\right) \left.\frac{dT^{-1}(Y)}{dY}\right|_{Y=y}$$

Considering the case where T is decreasing leads to the general result (1).

As an example, consider the reaction *spread* with rate law $f_{spread}(k_s, I, S, R) = k_s \cdot I \cdot S$. In the initial state, the rate of that reaction is $r = 50k_s$, where the pdf of k_s is uniform over $[0, 1]$. Applying (1) then gives the intuitive result that r is uniformly distributed over $[0, 50]$.

Once the stochastic relation has been constructed, it is then easy to build a PCMC that captures its behaviour by appropriately defining the constraint function. We omit the full details here but, intuitively, the states of the chain correspond to the systems that are "reachable" from the initial state of the model. The central part is the definition of the constraint function, for which we need the probability density of the transition rates between every pair of states. The latter are given directly by the stochastic relation. Note that there can be more than one reaction that leads to the same transition in the state space; the total transition rate is then the sum of the rates of the individual reactions, and the pdf of the sum of random variables is the convolution of their individual pdfs. For a system with k states, then, the constraint function ϕ over the rate r of transitioning from state s to state s' is given by:

$$\phi(s, s', r) = \bigotimes_{\alpha} f_\alpha(s')(r) \tag{2}$$

where $f_\alpha(s')(\cdot)$ is a pdf over the values of the transition rate from s to s' via reaction α (as given by the stochastic relation) and $\bigotimes$ denotes the convolution over the different reactions, as described above.

We briefly note that, in addition to this stochastic semantics, it is possible to define a continuous view of ProPPA models, as in [50]. We do not present the details here but will make use of such a continuous interpretation in the next section.

4.1 Concretization

We mentioned above that a PCMC can be used to formalise systems where transition rates are not fully specified. However, there are two ways of interpreting this uncertainty, first explored in [48] for discrete-time systems and more recently in [7] for the continuous-time case. Essentially, the difference is whether we allow the value of a particular rate to change during a run of the system. In the first case, we assume that rates can indeed change, so that each time we visit a state we must choose new values for them. This is referred to as a Markov Decision Process (MDP) semantics in [48]. Alternatively, under the Uncertain Markov Chain (UMC) semantics, we assume that each probability has a constant (but unknown) value. In this case, the values are fixed before the simulation and maintained throughout. In [7], the terms *uncertain* and *imprecise* are used respectively to distinguish the two settings. The ProPPA semantics can accommodate both interpretations; for the experiments presented here, we assume the uncertain scenario.

5 PROPPA TOOL SUITE

We have shown (Section 4) how the prior beliefs about the parameters induce a distribution over rates, but we have so far assumed that no observations are present. This section describes how the information from observations is incorporated into the model in order to obtain an updated distribution over the parameters. The ProPPA framework includes a range of inference methods for this task, detailed here, from which the user can choose and which can be automatically applied to the model. These include new implementations of algorithms for which no user-friendly tool exists to our knowledge, as well as newly-proposed methods.

5.1 Observations and inference

Observations represent new information which can affect our belief about how likely different values are. Inference can then be thought of as a transformation of the context, which takes the

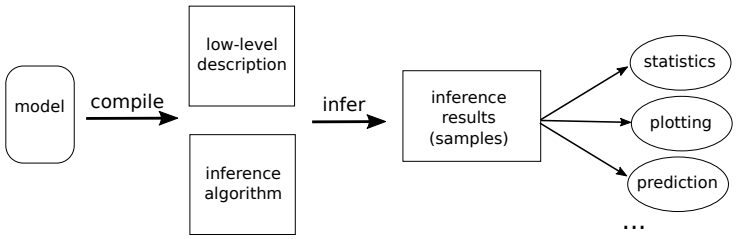

Fig. 3. Diagram of the different steps for inferring the parameters of a ProPPA model: the model is compiled to extract a low-level representation of the system, the chosen inference algorithm is applied and returns a set of samples from the posterior parameter distribution.

observations into account and updates the distributions of the uncertain parameters accordingly. From a probabilistic point of view, this corresponds to conditioning the prior distribution $p(\theta)$ on the observed data D, obtaining the *posterior distribution* $p(\theta \mid D)$. The relation between these quantities is given by Bayes' Theorem:

$$p(\theta \mid D) \propto p(\theta)p(D \mid \theta) \tag{3}$$

where $p(D \mid \theta)$ is the likelihood of the data, a measure of how likely we are to see these observations for a particular assignment of values to the parameters.

This view does not give any information on *how* to perform inference, however. Indeed, it is generally not possible to calculate the likelihood or the posterior analytically, so approximations must be used. The specification of the language allows for some freedom in the inference implementation; this approach creates a modular framework that can employ different algorithms, some examples of which are given later in this section.

In line 10 of Figure 2, the `observe` statement refers to an external file which holds our observations of the system. The `infer` statement on Line 11 specifies what inference algorithm will be used.

5.2 Overview of inference

The ProPPA framework has been implemented in Python and allows the analysis of a model through a number of algorithms. The remainder of this section focuses on how parameter inference is performed rather than what further analysis is possible. Processing a model includes various steps, summarised in the diagram of Figure 3.

Compilation. The text file containing the ProPPA model is first parsed and compiled into a lower-level representation. This is essentially a list of reactions, with their associated rates as executable functions; these functions accept a parameterisation and a state (vector of integers) and return a numerical rate. Additionally, an update vector is extracted for each reaction, and these are then combined into a stoichiometry matrix which shows how the quantity of each species is affected by each reaction. The appropriate prior distributions for each uncertain parameter are also generated. The observations file is loaded and interpreted as a time-series of measurements. Other necessary information, such as the chosen inference algorithm, is extracted from the model at this stage. This information is also sufficient to retrieve the state-space of the process (assuming it is finite); this may be required by some of the inference methods.

Inference. The different algorithms are applied to the low-level representation obtained as described above. The appropriate tool, as specified by the `infer` statement, is initialised using this representation and according to some default settings. These can be overridden if desired by using

an optional `configure` statement: this specifies a file with options that parameterize the algorithm itself; the available options vary according to the chosen algorithm. The inner workings of each algorithm vary and make use of different basic tools, including matrix exponentiation, stochastic simulation and solution of differential equations. The details of the various algorithms that form the core inference engine are presented in Section 5.3, along with their strengths and weaknesses; the decision of which to use depends on the comparative benefits of each method.

Output. Even though the prior distributions of the parameters are specified analytically, it is generally not possible to give an analytical expression for the posterior distribution. Instead, most of the algorithms we include are *Markov Chain Monte Carlo* (MCMC) methods that work via *sampling* a distribution. In other words, the output of the algorithm is a set of parameter values which can be used to reconstruct the posterior empirically or to compute quantities of interest, such as moments. MCMC is a standard tool in Machine Learning and Statistics, with a rich body of relevant research; for more information, we refer the interested reader to [29].

The simplest MCMC method is the *Metropolis-Hastings* (MH) sampler, which performs a random walk over the parameter space. In each iteration of MH, starting from a parameter value $\boldsymbol{\theta}$, a new value $\boldsymbol{\theta}^*$ is proposed. In our framework, $\boldsymbol{\theta}^*$ is sampled from a Gaussian distribution centred on $\boldsymbol{\theta}$, whose variance can be specified in the configuration file. An acceptance ratio a is then calculated to decide whether to move to the proposed $\boldsymbol{\theta}^*$ or remain at $\boldsymbol{\theta}$:

$$a = \frac{p(\boldsymbol{\theta}^*)}{p(\boldsymbol{\theta})} \frac{p(D \mid \boldsymbol{\theta}^*)}{p(D \mid \boldsymbol{\theta})} \tag{4}$$

where D are the observations. With probability $\min(a, 1)$, the next sample will be $\boldsymbol{\theta}^*$, otherwise it will be $\boldsymbol{\theta}$. This means that, for every sample taken, we need to compute the likelihood $p(D \mid \boldsymbol{\theta}^*)$. This calculation is not always possible, and the methods we offer are based on different ways of approximating or circumventing it. Note that by fixing the parameters, we essentially obtain a standard CTMC, which can be treated in the different ways laid out below.

The sequence of parameter samples that are returned by the algorithm are saved to an output file. These results can then be used to, for instance, produce histograms illustrating the posterior distribution, as shown in the examples of the next section. They can also form the basis of further analysis: for example, predictions of the system's behaviour, conditioned on the observations, can be made by choosing a value from the posterior samples, simulating the model for this parameter value, and repeating this process. This is outside the scope of the current version of the framework, however.

5.3 Inference algorithms

Within the ProPPA framework, we offer a suite of methods with complementary strengths and weaknesses, suitable to different models. Moreover, the framework is open and extensible, meaning that new methods can be incorporated as they are developed. We now present the methods which form the core inference engine of the language and highlight any restrictions they have; an important distinction is made according to whether a method can cope with systems with infinite state-spaces. These characteristics are summarised in Table 1.

5.3.1 Direct solution. The simplest algorithm we include works by direct computation of the likelihood, which is feasible (at least theoretically) if the state-space of the system is finite. It is based on a standard result for the transient solution of a CTMC: if the initial probability over states is $\mathbf{p}(0)$, then the probability at time t can be calculated via

$$\mathbf{p}(t) = \mathbf{p}(0)e^{\mathbf{A}t} \tag{5}$$

Table 1. Summary of the available algorithms in the inference engine.

Algorithm	Exact	Infinite	Notes
`direct`	Yes	No	Not suitable for large systems
`gibbs`	Yes	No	Requires Gamma priors and unique updates
`rouletteMH`	Yes	Yes	
`rouletteGibbs`	Yes	Yes	Requires Gamma priors and unique updates
`ABC`	No	Yes	Sensitive to chosen distance threshold
`fluid`	No	Yes	More accurate for systems with large counts
`LNA`	No	Yes	More accurate for systems with large counts, requires differentiable functions

where $\mathbf{A}$ is the infinitesimal generator matrix. The latter can be computed, for fixed values of the parameters, by simple processing of the representation extracted during compilation: for a starting state s and a reaction r, find the target state s' to which the process will jump if r occurs (by looking at the stoichiometry matrix), and compute the rate of that jump. This gives an entry $A_{s,s'}$ of the generator matrix.

Equation (5) can be used to compute the likelihood by repeated application. The state is observed at a finite number of time points t_i, and the measurements may be corrupted by noise. Assuming that the probability of observing state y when the state is actually x is $q(y \mid x)$, the likelihood of an observed time-series $D = ((t_1, y_1), \ldots, (t_N, y_N))$ is

$$p(D \mid \boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i, t_i \mid y_{i-1}, t_{i-1}, \boldsymbol{\theta}) \tag{6}$$

where (dropping the dependence on time for simplicity):

$$p(y_i \mid y_{i-1}, \boldsymbol{\theta}) = \sum_j p(x_i \mid y_{i-1}, \boldsymbol{\theta}) q(y_i \mid x_i) \tag{7}$$

Each term $p(x_i \mid y_{i-1}, \boldsymbol{\theta})$ can be calculated via (5). Since the likelihood can be computed, it is then possible to use a MH scheme to sample from the exact posterior of the parameters. This method of calculating the likelihood scales badly with the number of states because of the exponentiation of the generator matrix, therefore this method is only advisable for small examples.

5.3.2 Gibbs sampling. Another method included in our framework is based on Gibbs sampling, a special case of MH which improves the random walk behaviour and avoids the need to compute the likelihood. This means that it does not require matrix exponentiation, which makes it more suitable for models with larger spaces — it is still only applicable to finite systems, however. This method is based on the idea from [42], which is concerned with inferring the exit rates and jump probabilities of a CTMC conditioned on an observed time-series. We have modified it by adapting it to ProPPA-style descriptions of systems, so that we draw samples from the distribution of the model parameters. The improved efficiency of this method comes at a cost to its applicability: in order for Gibbs sampling to work, the kinetic law of each reaction must have the form $k_i f_i(s)$ where k_i is a parameter with a Gamma distribution and $f_i(s)$ is an arbitrary function of the state. Note that this includes, but is not limited to, mass-action laws. It is additionally required that each reaction have a distinct update vector.

5.3.3 Approximate Bayesian Computation. ProPPA can model systems of different size, including ones with infinite state-space. In the latter case, the two previous methods cannot be used; even for

large finite spaces, their performance can be unsatisfactory. One of the methods we include works around this limitation by following the Approximate Bayesian Computation (ABC) approach [49]. ABC requires only simulating the system, which makes it a good fit for CTMCs. For every parameter sampled, instead of computing the likelihood, the system is simulated via Gillespie's algorithm (which is straightforward to do from the extracted representation). The trace obtained is then compared to the observations using a distance metric. If this distance is greater than a threshold ϵ (specified by the user in the configuration file), the sample is rejected; otherwise, it is accepted with a probability similar to the one in (4). The parameter samples returned are drawn from an approximation to the posterior, one that converges to the true distribution as $\epsilon \to 0$. The accuracy of ABC depends on the value chosen for ϵ, but its simplicity means that it does not impose any constraints on the model and can be employed in cases where other algorithms fail.

5.3.4 Methods based on random truncation. In addition to ABC, the ProPPA framework includes two novel algorithms that can cope with infinite-state systems. These are implementations of methods we have recently developed [20] based on the idea of randomly truncating the state-space, applied to a high-level language for the first time. Instead of using a very large state-space, to accommodate for potential growth in the counts of the species, these methods work by choosing random truncations of the state-space in a way that ensures statistical correctness of the results. The two methods correspond to applying this idea to the direct and Gibbs sampling method described above. We briefly describe the changes, but for the full details refer the reader to [20].

Metropolis-Hastings. For this method, referred to as `rouletteMH`, the key idea is to express the likelihood as an infinite sum, and then truncate it according to the *Russian Roulette* [30] methodology. This gives an estimate $\hat{L}$ of the true likelihood that includes some error but is *unbiased*, i.e. $\mathbb{E}[\hat{L}] = P(D \mid \boldsymbol{\theta})$. This is done using an expansion of the transition probability so that the estimate can be computed in steps, corresponding to progressively larger (but finite) state-spaces. The estimate can then be used in the place of the true likelihood in the MH step. As the theory of pseudomarginal methods [3] shows, as long as $\hat{L}$ is unbiased, the samples will still be drawn by the true posterior distribution. The method is computationally more expensive than the direct approach (due to more steps being required to calculate $\hat{L}$) , but guarantees statistical correctness for systems with infinite state-spaces.

Gibbs-like sampler. This algorithm starts from the same base as the Gibbs sampler, with the addition of an extra step in which an upper bound for the state-space is chosen randomly. With this choice, we can use the finite-space Gibbs sampler, and still obtain correct results. The additional step also incurs a cost in performance, however, because of the need to correct for the choice of truncation point.

5.3.5 LNA-based inference. The final methods included in the core inference engine are based on a continuous approximation of the stochastic dynamics: instead of considering integer counts of each species, we view the state as a vector of continuous quantities, which are rescaled versions of the counts (in a biochemical setting, for example, these would be the concentrations of the different species). We also make use of the Linear Noise Approximation (LNA) [51]: this assumes that the state of the system follows a normal distribution whose mean and variance evolve in time. This evolution is described by a system of ODEs. The likelihood can be computed by numerical solution of this system, starting from the final time and moving backwards, with boundary conditions appropriately constructed from the observations as shown in e.g. [16]. All of the quantities needed to construct and solve the ODEs are easily obtained by the stoichiometry matrix, the compiled rate functions of the model and their derivatives (automatically calculated based on the syntax of the

rate function). Due to the assumptions of the LNA, this inference method is more appropriate for models with high counts of species, although this does not mean it cannot be applied to smaller models as well. The framework also includes a simpler variant, based on a deterministic continuous approximation giving the expected behaviour of the system. This is referred to as the fluid sampler.

It is worth pointing out that the ProPPA model itself does not need to undergo any changes; it is simply given a different interpretation in terms of this continuous view. It is straightforward to construct the necessary ODEs automatically from the formal description of the model, something which would involve more effort if working directly with the stochastic process instead of a high-level language.

5.4 Performance and scalability

The performance of the inference process naturally depends on the particular model and algorithm chosen. As expected, more complex models will require longer execution times, although the exact dependence varies with the inference method. For most algorithms, the crucial factor is the size of the state-space. This can, for instance, make the direct solver prohibitively expensive. For the Gibbs and Gibbs-like samplers, the amount of available memory is also particularly important. When using a continuous approximation, what matters is only the number of species rather than their upper bounds, as it directly affects the number of ODEs to be solved. The LNA-based method, while more accurate than the fluid one, is significantly more expensive, due to the higher number of ODEs involved. Its performance also depends on the number of observations, as the ODE solver is reinitialised after each observation point for improved numerical stability, following [16]. In contrast, the ABC-based algorithm, which employs stochastic simulation, depends more on the number of reactions than the number of species.

As an example, the experiment in Section 6.2, which uses the direct solver, took approximately 18 minutes to collect 10000 samples on a laptop. The experiments in Sections 6.1 and 6.3 use approximate solvers and were much faster, requiring about 8 and 10 minutes respectively to collect 10 times as many samples.

6 EXAMPLES

In this section we present three small examples to illustrate the syntax and inference capabilities of the ProPPA framework[1]. The first two involve finite populations while the third has an infinite state-space, requiring different methods.

6.1 Rumour spreading

We start by considering the example of Figure 2. For this experiment, we simulated the system with $k_s = 0.5$, $k_r = 0.1$, and used ten points from the resulting trajectory as the input time-series. The chosen values have no particular significance, but they roughly match the scale of values considered in recent inference work for the same model [8]. We use ABC to gather 100,000 samples of the two parameters; the results are plotted in Figure 4.

We can see the distribution of k_r is narrow, peaking close to its true value. In contrast, the posterior of k_s covers a much broader range of values. Indeed, simulating the system for values of k_s in $[0.4, 1.0]$ shows that its behaviour does not change much in this range, validating our results. Even with this wide distribution, however, the posterior differs from the prior in that it assigns little or no probability to values of k_s under 0.4, which produce significantly different behaviour to the one in the input observations.

[1]The implementation, including all examples and code to replicate them, is available at https://github.com/ageorgou/ProPPA.

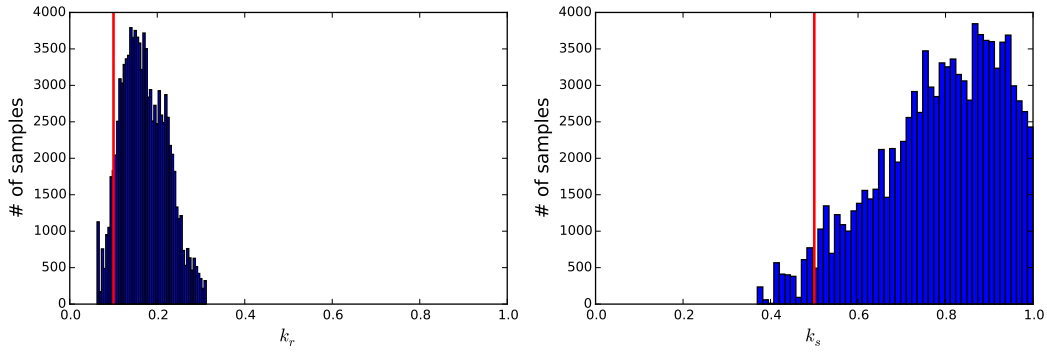

Fig. 4. Results from running ABC on the rumour-spreading example. The histograms show the accepted parameter samples for k_r (left) and k_s (right). The true values used to generate the data (0.1 and 0.5 respectively) are shown with red lines.

6.2 Epidemiological model

As a second example, we consider the simplest variant of the well-known SIR model, used in epidemiology to describe the spread of a disease. The model distinguishes between three types of individuals, here modelled as species: Susceptible, Infected and Recovered. Susceptible individuals become infected through contact with another infected person, and can recover without outside interference. We assume that all these interactions occur at rates given by mass-action laws.

```
r_i = Uniform(0,1);
r_r = Uniform(0,1);

kineticLawOf infect: r_i * S * I;
kineticLawOf recover: r_r * I;

S = infect ↓;
I = infect ↑ + recover ↓;
R = recover ↑;

S[10]<*>I[5]<*>R[0]

observe(obsSIR);
infer(direct);
```

Because of the structure of the system, it can be easily seen that the total population remains constant — individuals just move between the three states. An exact computation of the likelihood is therefore possible. Using the direct method described previously, we can sample from the posterior over parameters (Figure 5).

6.3 Predator-prey model

The third example describes a model often used in ecology to describe the interactions between a prey (X) and predator (Y). There are four reactions, representing the birth and the death of each species. The death rate of each species is proportional to its population, but in the case of the prey it also increases with the number of predators. Similarly, the rate at which predators are born depends

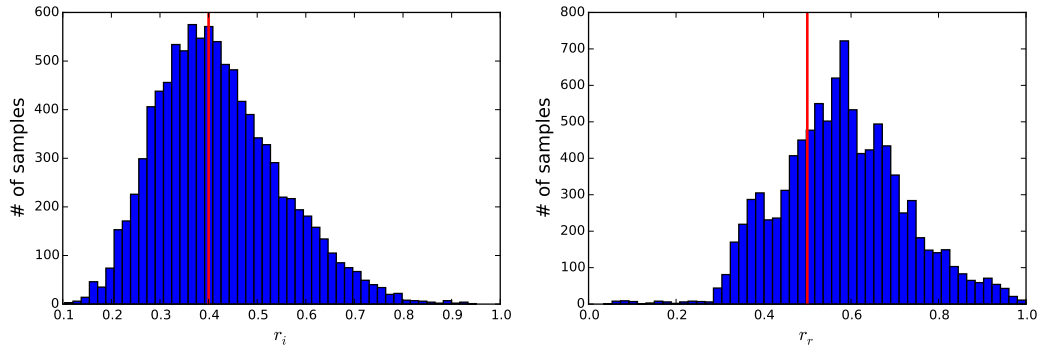

Fig. 5. Posterior samples for the infection (left) and recovery (right) rate in the SIR example using direct computation of the likelihood (true values: 0.4 and 0.5 respectively).

not only on their own count (number of possible parents), but also on that of the prey, accounting for the level of available sustenance. The underlying stochastic model is a Lotka-Volterra process, and the following is a description of the system in ProPPA:

```
a = Gamma(4,10000);
b = Gamma(4,10000);
c = Gamma(4,10000);
d = Gamma(4,10000);

kineticLawOf birthPred: a * X * Y
kineticLawOf deathPred: b * Y
kineticLawOf birthPrey: c * X;
kineticLawOf deathPrey: d * X * Y;

X = birthPrey ↑ + deathPrey ↓;
+ birthPred ⊕; //prey
Y = birthPred ↑ + deathPred ↓;
+ deathPrey ⊕; //predator

X[20]<*>Y[5]

observe(obsPredPrey);
infer(fluid);
```

In contrast with the previous example, this is an open system, as the count of either species can grow indefinitely. This excludes the use of the simpler algorithms, but methods like the fluid and LNA-based sampler are still applicable; in fact, another advantage of having the model formulated in a high-level, formal language is that the openness of the system can be detected statically and the user can be advised that their choice of algorithm is not appropriate (this kind of analysis can be performed in a similar way to other formalisms such as Petri Nets [34]). Note that, while the assumptions of these samplers make them more accurate for large populations, this does not preclude their use in this example. Switching to such a method simply requires changing the `infer` statement to refer to the fluid sampler, and making any adjustments to the configuration (e.g.

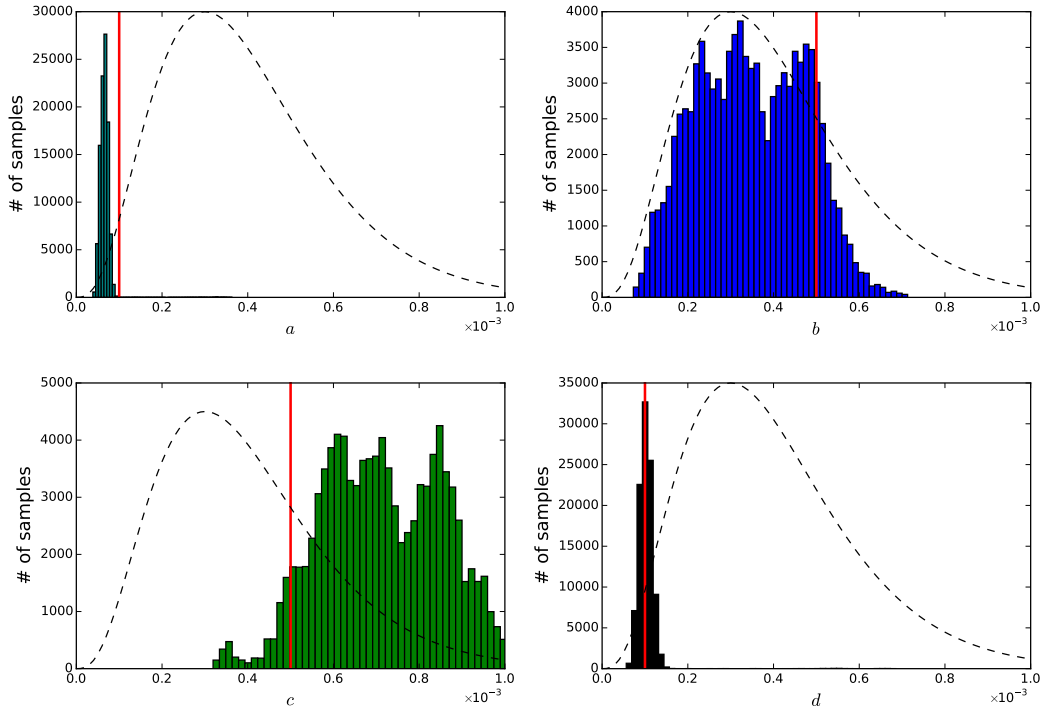

Fig. 6. Posterior samples for the parameters of the predator-prey model using the fluid sampler. True values used to generate the observations are 0.1×10^{-3} for a (teal) and d (black), and 0.5×10^{-3} for b (blue) and c (red). Prior shown in dashed line.

proposal distributions) that the user thinks are necessary. Figure 6 shows that the results are still accurate, with the exception of the parameter c, controlling the birth rate of the prey; a possible reason for this is that in the observations used the prey decline almost throughout, therefore the dataset is not particularly informative with respect to their birth rate. The results for b, the death rate of predators, could also indicate that the system is not particularly sensitive to its value as long as it remains within a certain range.

7 CONCLUSIONS AND FUTURE WORK

We have presented a process algebra that incorporates elements of probabilistic programming, the first such attempt at combining the two fields. This approach integrates uncertainty and observations into the system description, allowing us to model systems for which we have incomplete knowledge and giving us access to techniques from machine learning for inferring the unknown parameters.

The new features, while expanding the syntax of the language only minimally compared to a standard process algebra, significantly extend its expressive power. Additionally, our system is modular and flexible in the choice of inference algorithm, several of which are provided as part of the framework. Examples of their application on different models give promising results for the effectiveness of our approach.

We have focused on using time-series as observations, but an interesting next step would be to consider different types of observations, in particular using temporal logic formulae. Recent work [8]

has shown how to perform posterior inference for CTMCs based on a set of properties written in Metric Interval Temporal Logic. That method returns an analytical, albeit approximate, expression for the posterior over the parameters. We could apply a similar method to the examples considered here, by choosing logic properties that describe the observed behaviour well. Furthermore, the use of logic-based observations could enable additional analyses. We could explore, for example, how robust the behaviour of the system is with respect to high-level properties, in a similar spirit to recent work [21]. This would involve testing the satisfaction of given properties for different permissible values of the model's uncertain parameters. We leave this for future work.

Another interesting question for extending this work is whether the observations can be further integrated into the semantics of the language — for instance, whether the specification of a system can let us reject some transitions when building the underlying transition system. We are also interested in examining other benefits afforded by the use of a formal language, such as the description of equivalences and how they can be adapted to this probabilistic programming-like setting. Finally, we would like to expand our framework to integrate different analyses that could be performed, and offer an even more integrated workflow comprising model specification, estimation and analysis.

ACKNOWLEDGMENTS

This work was supported by Microsoft Research through its PhD Scholarship Programme. Jane Hillston and Anastasis Georgoulas acknowledge support from the EU FET-Proactive programme through QUANTICOL grant 600708. Guido Sanguinetti acknowledges support from the European Research Council through grant MLCS306999.

REFERENCES

[1] Alessandro Aldini, Marco Bernardo, and Flavio Corradini. 2010. *A process algebraic approach to software architecture design.* Springer.

[2] Waleed Alsanie and James Cussens. 2015. Learning failure-free PRISM programs. *International Journal of Approximate Reasoning* 67 (2015), 73 – 110. DOI:http://dx.doi.org/10.1016/j.ijar.2015.06.003

[3] Christophe Andrieu and Gareth O. Roberts. 2009. The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics* (2009), 697–725. http://www.jstor.org/stable/30243645

[4] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. 1996. Verifying continuous time Markov chains. In *Computer Aided Verification*, Rajeev Alur and ThomasA. Henzinger (Eds.). LNCS, Vol. 1102. 269–276. DOI:http://dx.doi.org/10.1007/3-540-61474-5_75

[5] Paolo Baldan, Andrea Bracciali, Linda Brodo, and Roberto Bruni. 2007. Deducing Interactions in Partially Unspecified Biological Systems. In *Algebraic Biology*, Hirokazu Anai, Katsuhisa Horimoto, and Temur Kutsia (Eds.). LNCS, Vol. 4545. 262–276. DOI:http://dx.doi.org/10.1007/978-3-540-73433-8_19

[6] Johannes Borgström, Andrew D. Gordon, Michael Greenberg, James Margetson, and Jurgen Van Gael. 2011. Measure transformer semantics for Bayesian machine learning. In *Proceedings of the 20th European conference on Programming languages and systems: part of the joint European conferences on theory and practice of software (ESOP'11/ETAPS'11).* Springer-Verlag, Berlin, Heidelberg, 77–96. http://dl.acm.org/citation.cfm?id=1987211.1987216

[7] Luca Bortolussi and Nicolas Gast. 2016. Mean Field Approximation of Uncertain Stochastic Models. In *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2016).*

[8] Luca Bortolussi and Guido Sanguinetti. 2013. Learning and Designing Stochastic Processes from Logical Constraints. In *Quantitative Evaluation of Systems*, Kaustubh Joshi, Markus Siegle, Marielle Stoelinga, and PedroR. D'Argenio (Eds.). LNCS, Vol. 8054. 89–105. DOI:http://dx.doi.org/10.1007/978-3-642-40196-1_7

[9] Richard Boys, Darren Wilkinson, and Thomas Kirkwood. 2008. Bayesian inference for a discretely observed stochastic kinetic model. *Statistics and Computing* 18 (2008), 125–135. Issue 2. DOI:http://dx.doi.org/10.1007/s11222-007-9043-x

[10] Luboš Brim, Milan Češka, Sven Dražan, and David Šafránek. 2013. Exploring parameter space of stochastic biochemical systems using quantitative model checking. In *Computer Aided Verification*. 107–123. http://link.springer.com/chapter/10.1007/978-3-642-39799-8_7

[11] Benoît Caillaud, BenoÃőt Delahaye, Kim G. Larsen, Axel Legay, Mikkel L. Pedersen, and Andrzej Wasowski. 2011. Constraint Markov Chains. *Theor. Comp. Science* 412, 34 (2011), 4373 – 4404. DOI:http://dx.doi.org/10.1016/j.tcs.2011.

05.010

[12] Laurence Calzone, Nathalie Chabrier-Rivier, François Fages, and Sylvain Soliman. 2006. Machine Learning Biochemical Networks from Temporal Logic Properties. In *Transactions on Computational Systems Biology VI*, Corrado Priami and Gordon Plotkin (Eds.). LNCS, Vol. 4220. 68–94. DOI: http://dx.doi.org/10.1007/11880646_4

[13] Federica Ciocchetta and Jane Hillston. 2009. Bio-PEPA: A framework for the modelling and analysis of biological systems. *Theor. Comp. Science* 410, 33-34 (2009), 3065–3084. DOI: http://dx.doi.org/DOI:10.1016/j.tcs.2009.02.037

[14] Edmund M. Clarke., E. Allen Emerson, and Aravinda Prasad Sistla. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.* 8, 2 (April 1986), 244–263. DOI: http://dx.doi.org/10.1145/5397.5399

[15] Daryl J Daley and David G Kendall. 1964. Epidemics and Rumours. *Nature* 204, 4963 (1964). DOI: http://dx.doi.org/10.1038/2041118a0

[16] Paul Fearnhead, Vasilieos Giagos, and Chris Sherlock. 2014. Inference for reaction networks using the linear noise approximation. *Biometrics* 70, 2 (2014), 457–466. DOI: http://dx.doi.org/10.1111/biom.12152

[17] Vashti Galpin. 2011. Equivalences for a biological process algebra. *Theor. Comp. Science* 412, 43 (2011), 6058 – 6082. DOI: http://dx.doi.org/10.1016/j.tcs.2011.07.006

[18] Anastasis Georgoulas, Jane Hillston, Dimitrios Milios, and Guido Sanguinetti. 2014. Probabilistic Programming Process Algebra. In *Quantitative Evaluation of Systems*, Gethin Norman and William Sanders (Eds.). Lecture Notes in Computer Science, Vol. 8657. Springer International Publishing, 249–264. DOI: http://dx.doi.org/10.1007/978-3-319-10696-0_21

[19] Anastasis Georgoulas, Jane Hillston, and Guido Sanguinetti. 2013. ABC-Fun: A Probabilistic Programming Language for Biology. In *Computational Methods in Systems Biology*, Ashutosh Gupta and ThomasA. Henzinger (Eds.). LNCS, Vol. 8130. 150–163. DOI: http://dx.doi.org/10.1007/978-3-642-40708-6_12

[20] Anastasis Georgoulas, Jane Hillston, and Guido Sanguinetti. 2016. Unbiased Bayesian inference for population Markov jump processes via random truncations. *Statistics and Computing* (2016), 1–12. DOI: http://dx.doi.org/10.1007/s11222-016-9667-9

[21] Mirco Giacobbe, Călin C. Guet, Ashutosh Gupta, Thomas A. Henzinger, Tiago Paixão, and Tatjana Petrov. 2016. Model checking the evolution of gene regulatory networks. *Acta Informatica* (22 Aug 2016). DOI: http://dx.doi.org/10.1007/s00236-016-0278-x

[22] Daniel T. Gillespie. 1977. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry* 81, 25 (1977), 2340–2361. http://pubs.acs.org/cgi-bin/archive.cgi/jpchax/1977/81/i25/pdf/j100540a008.pdf

[23] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: A language for generative models. In *UAI*. 220–229. http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.151.7160

[24] Noah D Goodman and Andreas Stuhlmüller. 2014. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org. (2014). Accessed: 2016-1-15.

[25] Andrew D. Gordon, Thore Graepel, Nicolas Rolland, Claudio Russo, Johannes Borgstrom, and John Guiver. 2014a. Tabular: A Schema-driven Probabilistic Programming Language. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '14)*. ACM, New York, NY, USA, 321–334. DOI: http://dx.doi.org/10.1145/2535838.2535850

[26] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. 2014b. Probabilistic Programming. In *Proceedings of the on Future of Software Engineering (FOSE 2014)*. ACM, New York, NY, USA, 167–181. DOI: http://dx.doi.org/10.1145/2593882.2593900

[27] Holger Hermanns. 2002. *Interactive Markov Chains: and the quest for quantified quality*. Springer-Verlag, Berlin, Heidelberg. http://dl.acm.org/citation.cfm?id=1744274

[28] Jane Hillston. 1996. *A Compositional Approach to Performance Modelling*. CUP. http://dl.acm.org/citation.cfm?id=236373

[29] Jun S. Liu. 2008. *Monte Carlo strategies in scientific computing*. Springer Science & Business Media.

[30] Anne-Marie Lyne, Mark Girolami, Yves Atchadé, Heiko Strathmann, and Daniel Simpson. 2015. On Russian Roulette Estimates for Bayesian Inference with Doubly-Intractable Likelihoods. *Statist. Sci.* 30, 4 (11 2015), 443–467. DOI: http://dx.doi.org/10.1214/15-STS523

[31] David Marco, David Cairns, and Carron Shankland. 2011. Optimisation of process algebra models using evolutionary computation. In *2011 IEEE Congress on Evolutionary Computation (CEC)*. 1296–1301. DOI: http://dx.doi.org/10.1109/CEC.2011.5949765

[32] David Marco, Erin Scott, David Cairns, Andrea Graham, Judi Allen, Simmi Mahajan, and Carron Shankland. 2012a. Investigating Co-infection Dynamics through Evolution of Bio-PEPA Model Parameters: A Combined Process Algebra and Evolutionary Computing Approach. In *Computational Methods in Systems Biology*, David Gilbert and Monika Heiner (Eds.). Lecture Notes in Computer Science, Vol. 7605. Springer Berlin Heidelberg, 227–246. DOI: http://dx.doi.org/10.1007/978-3-642-33636-2_14

[33] David Marco, Carron Shankland, and David Cairns. 2012b. Evolving Bio-PEPA process algebra models using genetic

programming. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference (GECCO '12)*. New York, NY, USA, 177–184. DOI: http://dx.doi.org/10.1145/2330163.2330189

[34] J. Martínez and M. Silva. 1982. *A simple and Fast Algorithm to Obtain all Invariants of a Generalised Petri Net.* Springer Berlin Heidelberg, Berlin, Heidelberg, 301–310. DOI: http://dx.doi.org/10.1007/978-3-642-68353-4_47

[35] Mieke Massink, Diego Latella, Andrea Bracciali, Michael D Harrison, and Jane Hillston. 2012. Scalable context-dependent analysis of emergency egress models. *Formal Aspects of Computing* 24, 2 (2012), 267–302.

[36] Tom Minka, John M. Winn, John P. Guiver, and David A. Knowles. 2012. Infer.NET 2.5. (2012). Microsoft Research Cambridge. http://research.microsoft.com/infernet.

[37] Rocco de Nicola, Diego Latella, Michele Loreti, and Mieke Massink. 2013. A Uniform Definition of Stochastic Process Calculi. *ACM Comput. Surv.* 46, 1, Article 5 (Oct. 2013), 35 pages. DOI: http://dx.doi.org/10.1145/2522968.2522973

[38] Aditya V. Nori, Chung-Kil Hur, Sriram K. Rajamani, and Selva Samuel. 2014. R2: An Efficient MCMC Sampler for Probabilistic Programs. In *AAAI Conference on Artificial Intelligence (AAAI)*. AAAI. http://research.microsoft.com/apps/pubs/default.aspx?id=211941

[39] Manfred Opper and Guido Sanguinetti. 2008. Variational inference for Markov jump processes. In *Advances in Neural Information Processing Systems 20*, J.C. Platt, D. Koller, Y. Singer, and S. Roweis (Eds.). MIT Press, Cambridge, MA, 1105–1112.

[40] Avi Pfeffer. 2007. The Design and Implementation of IBAL: A General-Purpose Probabilistic Language. In *Introduction to Statistical Relational Learning*, Lise Getoor and Ben Taskar (Eds.). The MIT Press. http://ieeexplore.ieee.org/xpl&arnumber=6278205

[41] Avi Pfeffer. 2009. CTPPL: A Continuous Time Probabilistic Programming Language. In *IJCAI*. 1943–1950.

[42] Vinayak Rao and Yee Whye Teh. 2013. Fast MCMC sampling for Markov jump processes and extensions. *Journal of Machine Learning Research* 14 (2013), 3207–3232. http://dl.acm.org/citation.cfm?id=2567768 arXiv:1208.4818.

[43] Aurélien Rizk, Gregory Batt, François Fages, and Sylvain Soliman. 2009. A general computational method for robustness analysis with applications to synthetic gene networks. *Bioinformatics* 25, 12 (2009). DOI: http://dx.doi.org/10.1093/bioinformatics/btp200

[44] Aurélien Rizk, Grégory Batt, François Fages, and Sylvain Soliman. 2011. Continuous valuations of temporal logic specifications with applications to parameter optimization and robustness measures. *Theoretical Computer Science* 412, 26 (2011), 2827 – 2839. DOI: http://dx.doi.org/10.1016/j.tcs.2010.05.008 Foundations of Formal Reconstruction of Biochemical Networks.

[45] Brian J Ross and Janine Imada. 2009. Evolving stochastic processes using feature tests and genetic programming. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM, 1059–1066. http://dl.acm.org/citation.cfm?id=1570044

[46] Taisuke Sato and Yoshitaka Kameya. 1997. PRISM: a language for symbolic-statistical modeling. In *In Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAIâĂŹ97*. 1330–1335.

[47] Eva Sciacca, Salvatore Spinella, Cristina Calcagno, Ferruccio Damiani, and Mario Coppo. 2013. Parameter Identification and Assessment of Nutrient Transporters in AM Symbiosis through Stochastic Simulations. *ENTCS* 293, 0 (2013), 83 – 96. DOI: http://dx.doi.org/10.1016/j.entcs.2013.02.020 Proceedings of CS2Bio'12.

[48] Koushik Sen, Mahesh Viswanathan, and Gul Agha. 2006. Model-Checking Markov Chains in the Presence of Uncertainties. In *Tools and Algorithms for the Construction and Analysis of Systems*, Holger Hermanns and Jens Palsberg (Eds.). LNCS, Vol. 3920. 394–410. DOI: http://dx.doi.org/10.1007/11691372_26

[49] Tina Toni, David Welch, Natalja Strelkowa, Andreas Ipsen, and Michael P.H Stumpf. 2009. Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of The Royal Society Interface* 6, 31 (2009), 187–202. DOI: http://dx.doi.org/10.1098/rsif.2008.0172

[50] Mirco Tribastone, Stephen Gilmore, and Jane Hillston. 2012. Scalable Differential Analysis of Process Algebra Models. *Software Engineering, IEEE Transactions on* 38, 1 (Jan 2012), 205–219. DOI: http://dx.doi.org/10.1109/TSE.2010.82

[51] Nicolaas Godfried Van Kampen. 1992. *Stochastic processes in physics and chemistry*. Elsevier.

[52] Ludovica Luisa Vissat, Allan Clark, and Stephen Gilmore. 2015. Finding Optimal Timetables for Edinburgh Bus Routes. *Electronic Notes in Theoretical Computer Science* 310 (2015), 179 – 199. DOI: http://dx.doi.org/10.1016/j.entcs.2014.12.018 Proceedings of the Seventh International Workshop on the Practical Application of Stochastic Modelling (PASM).

[53] Christopher D. Williams and Jane Hillston. 2014. Automated Capacity Planning for PEPA Models. In *Computer Performance Engineering*, András Horváth and Katinka Wolter (Eds.). Lecture Notes in Computer Science, Vol. 8721. Springer International Publishing, 209–223. DOI: http://dx.doi.org/10.1007/978-3-319-10885-8_15

A FULL SEMANTICS OF PROPPA

The construction of the capability and stochastic relations (Section 4) was first presented in [18].

The rules governing the relations are given in the FuTS [37] style, an alternative formalism to the more common small-step style semantics. In FuTS, transitions are specified using a *continuation*, a function over possible targets. The codomain of the continuation differs depending on the kind of transition (e.g. deterministic, stochastic, ...). The shorthand notation $[s_1 \mapsto v_1, \ldots, s_n \mapsto v_n]$ denotes a continuation assigning the value v_i to each s_i, and the zero value of its codomain to each other state $s \neq s_1, \ldots, s_n$. For instance, a non-deterministic transition from state s to either s' or s'', but no other possible target, would be represented as $s \rightarrowtail [s' \mapsto \mathit{true}, s'' \mapsto \mathit{true}]$. If the two target states had associated probabilities, then those would be the values of the continuation instead (with every other state being assigned a value of 0).

A.1 Capability relation

In ProPPA, as in Bio-PEPA, the capability relation is qualitative: it expresses only whether a transition is possible and does not give any information about its rate. It is therefore, in a way, of "structural" concern, a consequence purely of the species definitions. It is the smallest relation that satisfies the rules given in Figure 7.

The first three rules, `PrefixReac`, `PrefixProd` and `PrefixMod`, define the behaviour of a simple component that is a reactant, product or modifier, respectively, in a single reaction. Such a component can only perform one action, as indicated by the "singleton" notation $[S \mapsto \mathit{true}]$ in these rules. The transition label contains two elements: the reaction name, and a list of roles. Each element of this list has the form $S : op(l, k)$, with $op \in \{\downarrow, \uparrow, \oplus, \ominus, \odot\}$ and $l, k \in \mathbb{N}$. This indicates that species S has role op and an initial level of l, which will change by k.

The rules `Choice1` and `Choice2` formalise the meaning of the choice operator +. As mentioned above, a component $P_1 + P_2$ can perform all actions that either P_1 or P_2 can perform.

The semantics of the cooperation operator is defined in the rules `Coop1`, `Coop2` and `Coop3`. For the behaviour of $P_1 \underset{L}{\bowtie} P_2$ when considering an action α, we make a distinction according to whether or not α is shared between the cooperating components. If it is not, but one of them can perform α, then the other component is ignored and the $P_1 \underset{\alpha}{\bowtie} P_2$ behaves as P_1 or P_2 (rules `Coop1`, `Coop2`). If α is shared, however, both components participate in the action, and we must record the changes for both of them. This means concatenating the labels from both of their individual transitions, indicated by the :: notation (`Coop3`).

A.2 Stochastic relation

As mentioned in the main text, the stochastic relation describes the uncertainty over the rate of a transition.

In contrast to the reachability information captured by the capability relation, deriving the rates requires additional elements of the model — namely, the kinetic laws and the parameter definitions. The stochastic relation is thus defined over entire models rather than components. It builds on the capability relation, using the information contained therein; specifically, the state, which is contained in the capability relation labels, is passed on to the appropriate kinetic law. Using the FuTS notation, $\mathcal{M} \overset{\alpha}{\rightarrowtail}_s f$ with $f(\mathcal{M}') = \mu$ means that $\mathcal{M}$ can transition to $\mathcal{M}'$ via reaction α, and the rate of that transition is distributed according to μ. A fixed rate r is expressed by the Dirac distribution $\delta(r)$, where all the probability mass is assigned to r. The degenerate case $\delta(0)$ indicates that the transition occurs at rate 0, i.e. never; this is used for completeness, when we need to describe transitions that are not allowed by the capability relation.

The stochastic relation is the smallest relation for which the following rule holds:

$$\frac{P \overset{(a,w)}{\rightarrowtail}_c g}{\langle \mathcal{T}, P\rangle \overset{a}{\rightarrowtail}_s h_{g,w,\mathcal{T}}}$$

where the function h maps models to distributions of rates:

$$h_{g,w,\mathcal{T}}(\langle \mathcal{T}', s'\rangle) = \begin{cases} \delta(0) & \text{if } \mathcal{T}' \neq \mathcal{T} \\ \delta(0) & \text{if } g(s') = \mathit{false} \\ \mu & \text{otherwise} \end{cases}$$

Essentially, h makes sure that the capability relation is respected by assigning zero rates to unreachable states or different contexts. The distribution μ over rates in the non-trivial cases is obtained through appropriate variable transformations, as explained in Section 4.

Having access to the stochastic relation, a ProPPA model $\langle \mathcal{T}, P\rangle$ can further be mapped to a PCMC $\langle S, o, A, V, \phi\rangle$, where:

- $o = P$, the model component of the model,
- $S = \boldsymbol{ds}(\langle \mathcal{T}, P\rangle)$,
- $A = \varnothing$ and $V(s) = \{\varnothing\}\ \forall s \in S$,
- $\phi(s, s', r) = \bigotimes_\alpha f_\alpha(s')(r)$, the distribution of the total rate of transitioning from s to s', as explained for Equation (2).

$$\begin{array}{llll} \texttt{PrefixReac} & (a,k) \downarrow S(l) & \overset{(a,[S:\downarrow(l,k)])}{\rightarrowtail}_c & [S(l-k) \mapsto \mathit{true}] \quad l \geq k \\ \texttt{PrefixProd} & (a,k) \uparrow S(l) & \overset{(a,[S:\uparrow(l,k)])}{\rightarrowtail}_c & [S(l+k) \mapsto \mathit{true}] \quad l \geq 0 \\ \texttt{PrefixMod} & (a,k)\ op\ S(l) & \overset{(a,[S:op(l,0)])}{\rightarrowtail}_c & [S(l) \mapsto \mathit{true}] \quad l \geq k \end{array}$$

$$\texttt{Choice1}\ \frac{P_1 \overset{(a,w)}{\rightarrowtail}_c f}{P_1 + P_2 \overset{(a,w)}{\rightarrowtail}_c f} \qquad \texttt{Choice2}\ \frac{P_2 \overset{(a,w)}{\rightarrowtail}_c f}{P_1 + P_2 \overset{(a,w)}{\rightarrowtail}_c f}$$

$$\texttt{Coop1}\ \frac{P_1 \overset{(a,w)}{\rightarrowtail}_c f_1 \qquad a \notin \mathcal{L}}{P_1 \underset{\mathcal{L}}{\bowtie} P_2 \overset{(a,w)}{\rightarrowtail}_c g_1} \qquad \texttt{Coop2}\ \frac{P_2 \overset{(a,w)}{\rightarrowtail}_c f_2 \qquad a \notin \mathcal{L}}{P_1 \underset{\mathcal{L}}{\bowtie} P_2 \overset{(a,w)}{\rightarrowtail}_c g_2}$$

$$\texttt{Coop3}\ \frac{P_1 \overset{(a,w_1)}{\rightarrowtail}_c f_1 \qquad P_2 \overset{(a,w_2)}{\rightarrowtail}_c f_2 \qquad a \in \mathcal{L}}{P_1 \underset{\mathcal{L}}{\bowtie} P_2 \overset{(a,w_1 :: w_2)}{\rightarrowtail}_c g}$$

where:

$$g_1(Q) = \begin{cases} f_1(Q_1) & \text{if } Q = Q_1 \underset{\mathcal{L}}{\bowtie} P_2 \\ \mathit{false} & \text{otherwise} \end{cases} \qquad g_2(Q) = \begin{cases} f_2(Q_2) & \text{if } Q = P_1 \underset{\mathcal{L}}{\bowtie} Q_2 \\ \mathit{false} & \text{otherwise} \end{cases}$$

$$g_1(Q) = \begin{cases} f_1(Q_1) \wedge f_2(Q_2) & \text{if } Q = Q_1 \underset{\mathcal{L}}{\bowtie} Q_2 \\ \mathit{false} & \text{otherwise} \end{cases}$$

$$\texttt{Constant}\ \frac{P \overset{(a,w)}{\rightarrowtail}_c f \qquad Q \overset{def}{=} P}{Q \overset{(a,w[P \to Q])}{\rightarrowtail}_c f}$$

Fig. 7. Capability relation semantic rules for ProPPA models expressed in the FuTS formalism.

Here, $\boldsymbol{ds}$ is the *derivative set* of a model $\mathcal{M}$, the set of all its reachable configurations. Formally, $\boldsymbol{ds}(\mathcal{M})$ is defined as the smallest set such that:

i. $\mathcal{M} \in \boldsymbol{ds}(\mathcal{M})$, and
ii. If $\mathcal{P} \in \boldsymbol{ds}(\mathcal{M})$ and, for some α, $\mathcal{P} \overset{\alpha}{\rightarrowtail}_s f$ with $f(\mathcal{P}') \neq \delta(0)$, then $\mathcal{P}' \in \boldsymbol{ds}(\mathcal{M})$

Note that, since we are not concerned with model checking, we provide no propositions or labellings. This construction corresponds to the Uncertain Markov Chain interpretation (Section 4.1).